Parte 1: Fundamentos da linguagem Java

Capítulo 1

Introdução à linguagem Java

Bom, agora que nós já conhecemos um monte de coisas a respeito da linguagem Java: como começou, suas características principais, como instalar e configurar um ambiente Java em Windows e Linux, os utilitários básicos do kit Java e também onde conseguir ajuda está na hora de começarmos a por a mão na massa e dar início a programação básica. No entanto seria bom já por a água para esquentar porque precisaremos de algumas xícaras de café. Começaremos aqui pelos fundamentos da linguagem e na seqüência entraremos na filosofia e orientação a objetos, que se você prestar atenção não vai ter muitas dificuldades.

Mas antes eu já vou lhe avisando, leia com muita atenção estes capítulos seguintes, estude e faça os exemplos, faça os exercícios propostos e se for necessário refaça o capítulo que não entendeu direito, pois na linguagem Java não dá para "enrolar", ou você sabe ou não sabe e eu prefiro que você saiba, afinal de contas, quer ser um profissional nessa área, certo?

Se você já programou em alguma linguagem já deve então saber que precisamos seguir algumas regras básicas para que nosso programa flua de acordo com nosso desejo e mesmo assim tem horas que não sai do jeito que a gente quer.

A linguagem Java tem uma estrutura de programação, digamos, semelhante com a estrutura das outras linguagens de programação, como declarações iniciais, regras para funções que aqui chamaremos de métodos, declarações e variáveis, operadores, etc, porém o conceito é bem diferente daquele que encontramos no *Visual Basic*, *Delphi,* etc. Na linguagem Java você trabalha com a idéia de objetos e esses objetos têm uma função no sistema que está sendo desenvolvido, e isso tudo aprenderemos quando começarmos a estudar Java com orientação a objetos.

Por enquanto vamos nos preocupar em entender o processo básico de programação no Java e eu já lhe digo, esse sim, é semelhante ao que você pode encontrar na linguagem C/C++, Pascal, etc. Se você pretende mais tarde tirar uma certificação em Java, fique sabendo que vai ter que estudar bastante, pois as provas exigem muito de seus candidatos. Portanto seja um bom leitor/aluno e estude bem os exemplos contidos neste livro, já vai ajudar bastante.

Estrutura de um programa em Java

Para que tenha uma idéia do que estou falando, vamos analisar um exemplo bem simples e básico de um pequeno programa em Java, nele percebemos os conceitos e as regras das quais eu lhe falei. Procure entender logo cedo para que todo o conceito seja fácil de ser assimilado, procure identificar quais são as partes de um programa estão "pedindo" um objeto, um método, uma classe, um atributo, etc. Se chegar perceber isso, você está no meio do caminho do sucesso, que legal isso hein?

Acompanhe a listagem de um programa para o modo texto, apenas lembrando que a numeração das linhas é apenas para interpretação, não fazendo assim parte de um programa Java.

```
1. /**
2.  The HelloWorldApp class implements an application that
3.  displays "Hello World!" to the standard output.
4.  */
5.  public class HelloWorldApp {
6.         public static void main(String[] args) {
7.          // Display "Hello World!"
8.          System.out.println("Hello World!");
9.         }
10.  }// End of program.
```

Vamos fazer uma análise desse pequeno programa:

— As linhas de 1 a 4 são comentários no código Java para auxiliar na interpretação e leitura do programa.

— A linha 5 declara uma classe chamada *HelloWordlApp*, ou seja, esse vai ser o nome do programa depois de compilado. Salve o arquivo obrigatoriamente com esse nome e seguindo a regra que nós já sabemos que o Java é *case sensitive*, ou seja, faz distinção entre letras maiúsculas e minúsculas, por exemplo, *ADILSON.java* é diferente de *Adilson.java*. O nome do arquivo salvo será *HelloWordlApp.java* e o nome do arquivo compilado será *HelloWordlApp.class*, mas tenha calma isso você vai aprender logo, logo.

DICA: No padrão sugerido pela SUN toda classe tem iniciais maiúsculas, todo método é em minúscula e declaração de objetos também, pois é válido a existência de um objeto com o mesmo nome da classe.

— A linha 6 é o método principal que todo programa Java tem exceto os *applets* que aprenderemos o por que quando estiver estudando o capítulo sobre Java *applets*. É como se fosse a função *main()* da linguagem C.

— A linha 7 é mais um modelo de comentário em Java.

— A linha 8 é responsável por imprimir a mensagem *Hello World!* na tela. É como o comando *printf* da linguagem C.

— A linha 9 encerra o método *main(...)* declarado na linha 6 e...

— E por último a linha 10 encerra a classe *HelloWordlApp* declarada na linha 5.

Fácil de entender esse programa, não é mesmo? Bom, acho que você entendeu e pode perceber que se começarmos com exemplos bem simples o Java acaba se tornando de certa forma até que "fácil".

O código anterior gera a frase "*Hello World!*" em MS-DOS ou no terminal do sistema Linux. Caso queira executá-lo, primeiramente salve o arquivo e execute o comando *javac HelloWordlApp.java* para compilar e gerar o arquivo *HelloWordlAp.class* agora para executar o programa digite o comando *java HelloWordlApp*. OK! Seu programa esta aí.

```
// Comentários iniciais sobre o programa.
package com.studyguide.myjava;                              //1

import com.unipar.cooperate.server.*;                       //2
import java.util.*;                                         //3
import javax.swing.*;
import java.awt.*;

public class AloMundao {                                    //4
      public final static void main(String[] args) {       //5
         System.out.println("Alo Mundao Java!");            //6
      }
} // fim do programa.
```

Resumindo, devemos saber em princípio que todos os programas em Java possuem 6 (seis) elementos básicos, os quais podem ou estaram presente em seus programas conforme o código anterior, onde:

(1) Pacote Java criado por terceiros. Esses pacotes podem incluir inúmeras funcionalidades a mais para dar mais poder as suas aplicações.

(2) Classes definidas por você mesmo ou classes de terceiros para aproveitamento de código. Por exemplo, uma classe para verificar CPF, CNPJ, RG, Título Eleitoral, Inscrição Municipal/Estadual e assim por diante.

(3) Pacotes do próprio kit SDK da linguagem Java.

(4) Classe principal a qual vai dar nome a sua aplicação. Todo programa em Java apresenta uma classe principal caso siga esse exemplo.

(5) Método principal o qual executará o programa. Todo programa em Java tem um método principal semelhante à linguagem C/C++ caso siga esse exemplo. O método *main()* é muito flexível, a ordem dos modificadores não importa, o argumento é do tipo array de Strings ou *varargs* (Java 5) e o nome do argumento pode ser qualquer nome válido.

```
static public void main(String[] args){...}
public static void main(String... x) {...} // Java 5.
static public void main(String xyz[]){...}
```

(6) Corpo do programa. Dentro do corpo do programa é que são usados variáveis, operadores, chamadas de métodos e classes (internas ou externas), outros programas, rotinas do sistema operacional, acesso a intranet e a Internet etc.

Você não necessariamente terá que usar os passos (1) e **(2)** para desenvolver seus programas em Java. Isso acontecerá com o tempo onde os programas ficam grandes e complexos.

Como podemos ver, essa é a regra para um programa linear, ou seja, um programa de um único arquivo *.java* que vai ser compilado e executado. Essa regra vale para o modo texto e também para o modo gráfico. Não se preocupe agora com programas modulares, como, um programa semelhante a idéia do Delphi onde temos vários arquivos de formulários que são partes de um programa.

A estrutura básica de programação em Java não é difícil de ser seguida, pois ela define as seguintes regras:

1. *Atribuição de nomes*: Se o arquivo fonte possuir uma classe pública, seu nome deverá ser o mesmo que o dessa classe;
2. *Total de classes*: A linguagem Java sugere o máximo de uma classe pública em um arquivo fonte. Porém o restante não sendo pública pode ter quantas classes o programador desejar, no entanto, a SUN recomenda o máximo de linhas por arquivo fonte seja em número de 2.000 (dois mil), para não afetar a performance;
3. *Layout*: Um arquivo fonte .java pode conter três elementos de "nível superior" que são:
    a) Uma declaração de *package* opcional;
    b) Qualquer número de declarações de importação e
    c) Declarações de classe e de interface. Lembre-se de que esses itens devem aparecer nessa ordem. Isto é, as instruções de importação (*import*) devem preceder todas as declarações de classe e, se uma declaração de *package* for utilizada, deverá preceder as classes e as importações (*import*).

Explanação da estrutura básica

Eu sei que você é observador de pode observar todas as partes desse programa pequeno e simples, mas com certeza teve algumas dúvidas como, sobre as palavras, instruções e comandos desse programa. Pois bem, vou lhe explanar aqui de uma forma bem clara:

1. *public*: É um quantificador do método que indica que ele é acessível externamente a esta classe (para outras classes que eventualmente seriam criadas). Uma classe pública pode ser vista por todo o programa;
2. *static*: É um qualificador que indica que o método ou variável deve ser compartilhado por todos os objetos que são criados a partir dessa classe. Os métodos *static* podem ser invocados, mesmo quando não for criado nenhum objeto para a classe, para tal deve-se seguir a sintaxe: <Classe>.<Método> (argumentos);
3. *void*: É semelhante ao *void* da linguagem C/C++, corresponde ao valor de retorno da função. Quando a função não retorna nenhum valor ela retorna *void* (vazio), uma espécie de valor vazio que tem que ser especificado;
4. *main()*: Semelhante ao *main()* da linguagem C/C++, um nome particular do método que indica para o compilador o início do programa, é dentro deste método e através de interações entre os atributos e argumentos visíveis que o programa se desenvolve. Chamamos de método principal que é o ponto de entrada para todo e qualquer programa e todo programa apresenta somente um desse método por arquivo;
5. *String args[]*: Esse é o argumento do método principal (*main*) e por conseqüência, do programa todo. Ele é um vetor de strings que é formado quando são passados ou não argumentos através da invocação do nome do programa na linha de comando do sistema operacional. Também poderemos encontrar assim: *String[] args;*
6. {....}: Delimitam um bloco de código, isso é igual a linguagem C/C++ e semelhante ao BEGIN e END em Pascal;
7. *System.out.println()*: É a chamada do método *println* para o atributo *out* da classe *System*. O argumento de *println* é uma constante do tipo String. O *println* assim como o *writeln* de Pascal, imprime uma string qualquer e posiciona o cursor na linha abaixo, analogamente o *print* não avança a linha.

No decorrer do seu aprendizado do Java você pode encontrar diferentes tipos de declaração para a classe *System* e para que você não fique a ver navios aqui estão os três tipos que você poderá encontrar:

1. *System.out*: Usada para a saída padrão (tela) de informações. Usamos para mostrar na tela informações de processamento dos nossos programas;
2. *System.in*: Usada para entrada padrão (teclado) de informações. Usamos quando precisamos que o usuário forneça informações ao sistema (aplicação);
3. *System.err*: Usada para saída padrão (arquivo ou tela) e erros (informações). Usamos quando precisamos analisar eventuais informações decorrentes de erros em nossos aplicativos.

DICA: Por padrão, todas as classes são subclasses da classe Object (classe mais alta de todas), mas não precisa se preocupar, pois isso já está implícito em todos os seus programas. É comum e você verá neste livro o uso dessa classe para verificação de objetos.

Agora acompanharemos outro exemplo de um programa que usa a mesma idéia, porém é para o modo gráfico que usa o *Swing* (pacote Java para escrever aplicações em modo gráfico). Perceba que a idéia é a mesma, ou seja, a de mostrar a mensagem *Hello World!* na tela:

```
1. import javax.swing.*;
2.
3. public class HelloWorldSwing {
4.       public static void main(String[] args)  {
5.
6.       JFrame frame = new JFrame("HelloWorldSwing");
7.       final JLabel label = new JLabel("Hello World");
8.       frame.getContentPane().add(label);
9.
10.       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11.
12.       frame.pack();
13.       frame.setVisible(true);
14.       }
15. }// Fim do programa.
```

Vamos analisar esse programa:

— A linha 1 importa do pacote Swing *javax.swing.*;* que é responsável pela parte gráfica (janelas, botões, etc), ou seja, todos os componentes para o desenho de nossa interface, assim não precisaremos nos preocupar com os nomes de tais componentes. O pacote *Swing* é melhor e mais moderno que o *AWT*.

— A linha 3 e 4 são declarações da classe e do método principal.

— A linha 6 informa a criação de um objeto chamado *frame* de uma classe *JFrame*. Isso é, uma janela completa (bordas e botões) com o título *HelloWorldSwing*.

— A linha 7 informa a criação de um objeto chamado *label* de uma classe chamada *JLabel* e ele vai colocar o texto *HelloWorldSwing* dentro da janela.

— A linha 8 insere o objeto *label* em um painel que está contido dentro do *frame*. É assim, uma janela (*frame*) apresenta um painel que é o *getContentPane()* e dentro desse painel inserimos todos os objetos que fazem parte da interface. A principal função dos painéis é que eles servem para controlar o posicionamento de seus componentes para uma melhor organização de sua aplicação.

— A linha 10 programa o botão X para fechar a janela. Por padrão ele está desabilitado e não vai fechar a janela quando você clicar nele, é por isso que você precisa dessa linha. Engraçado né? Após a versão 1.3 do Java somente essa linha basta para que o botão X tenha seu efeito de fechar a janela. Caso você use uma versão anterior ao 1.3 (o que eu não recomendo), você precisará criar um *listener* (ouvinte), que é algo semelhante aos eventos do Visual Basic e do Delphi, Assim ele ficará "ouvindo" se algo aconteceu e poderá então executar uma ação (*Action*).

— As linha 12 e 13 são responsáveis por disponibilizar a janela na tela. Há outros modos de se disponibilizar uma janela na tela mesmo usando AWT e Swing, porém com o tempo você aprenderá como fazer.

— As linhas 14 e 15 encerram a declaração da classe (linha 3) e do método principal (linha 4).

Veja que interessante, o código anterior gera a seguinte saída em Windows e em Linux:

Figura 1.1: Resultado do programa em Windows e Linux.

NOTA: Em Java nós programamos usando vários objetos e esses objetos ficam contidos em painéis os quais poderão estar contidos em outros painéis. Assim estamos usando a filosofia de camadas, onde cada uma tem as suas próprias regras.

Para exemplos simples, usualmente podemos também fazer uso das caixas de diálogos, onde essas possuem a vantagem de trazer botões já padronizados. Acompanhe o próximo exemplo:

```
import javax.swing.JOptionPane;

public class AloMundoJOP
{
      public static void main( String args[])       {
         JOptionPane.showMessageDialog(null,"Alo Mundo!\nBem Vindo ao Java.");
         System.exit(0); // Termina a aplicação.
      }
}// Fim do programa.
```

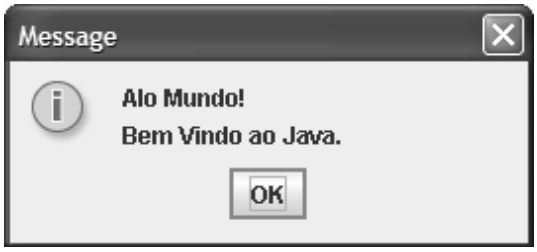

Figura 1.2: Resultado do programa em Windows e Linux.

DICA: Esses tipos de caixas de diálogos podem ser personalizadas para diversos tipos de mensagens no sistema, por exemplo, caixas de mensagens de erros (ERROR_MESSAGE), de informação (INFORMATION_MESSAGE), e de alerta (QUESTION_MESSAGE), mensagens de avisos (WARNING_MESSAGE), e de texto puro (PLAIN_MESSAGE). Também pode-se personalizar os títulos e os ícones e até mesmo os sons emitidos.

No sistema Linux devido aos vários gerenciadores de janelas a aparência de seu aplicativo muda bastante. Experimente usar os gerenciadores *KDE*, *Gnome*, *WindowMaker*, *IceWM,* etc para ver como é que fica a aparência de seus programas. Devido a essa filosofia do Linux nossos programas ficam muito mais bonitos e interessantes de usar.

Lembrando que não entraremos no conceito técnico da programação, pois nós somente estamos fazendo uma análise de um programa em código Java. Eu sei que você vai fazer a seguinte pergunta: *"— Nossa, tudo isso só para colocar uma mensagem na tela?"*.

Eu digo: *"— Só isso, e já pode ir se acostumando que você vai digitar um bocado para aprender a programar em Java"*. Em Java quando estamos digitando nossos programas "no braço", ou seja, sem contar com a ajuda de uma IDE (ambiente de desenvolvimento), usamos de muita digitação e, portanto é MUITO importante desde já que nos acostumemos a corrigir *on-time* (na hora), os nossos erros, pois se faltar uma vírgula ou ponto e vírgula (com exceção dos comentários todas as linhas de um programa em Java terminam com ponto e vírgula), teremos que procurar e isso poderá gerar certo aborrecimento pelo tempo que vai ser gasto. Pois dificilmente nas primeiras tentativas de programação a gente acerta.

Claro, que depois que você tiver já uma boa base e idéia do ambiente Java, você vai escolher um programa para lhe ajudar a digitar melhor e depois vai escolher uma IDE para daí começar a desenvolver suas aplicações cada vez com um nível maior e conseqüentemente uma maior complexidade.

Imagine um sistema de banco de dados, com validação de usuário, conexões remotas, gerenciamento de *backups*, emissão de relatórios em texto, em PDF, HTML, etc e etc. Você acha que isso é simples de ser feito? Pois, acredite, não é. Isso geralmente é e deve ser feito em equipe e dependendo da complexidade de um sistema pode levar muito tempo. Eu não estou querendo lhe desanimar, muito pelo contrário, estou lhe mostrando o quanto o Java tem para lhe oferecer e você para aprender, e assim tornar-se um profissional qualificado.

As principais classes e pacotes do Java

A linguagem Java é extremamente poderosa e possui inúmeras classes e métodos para podermos desenvolver nossos aplicativos de uma maneira bem mais rápida, flexível e robusta economizando tempo como faríamos na linguagem C/C++ através do uso de bibliotecas e métodos.

Se fizermos uma analogia com C/C++, as classes em Java são semelhantes às bibliotecas e os métodos são semelhantes às funções e do mesmo modo que podemos criar nossas próprias funções e bibliotecas na linguagem C/C++, também podemos fazer isso na linguagem Java.

Porém na linguagem Java nós temos os pacotes, que são arquivos contendo classes, interfaces e métodos para serem usados na linguagem. Os pacotes na linguagem Java devem ser os primeiros a serem declarados logo no início do código.

Os pacotes padrão da linguagem Java que são importados para dentro de um código ou aplicação tem a seguinte sintaxe:

```
import javax.swing.*;         // Tudo que for Swing.
import java.awt.*;            // Tudo que for AWT.
```

Também podemos fazer importações específicas para somente o que nos interessa, veremos essas importações no capítulo sobre coleções.

```
import java.awt.Graphics;             // Importa Graphics.
import javax.swing.JOptionPane;       // Importa JoptionPane.
```

Os pacotes de terceiros ou os que você mesmo criar tem uma outra forma de serem declarados e vêem antes do *import* para pacotes normais, ou seja, deve ser a primeira linha a ser declarada. Podem existir apenas comentários antes dessa declaração. Embora aqui apareçam vários pacotes, apenas uma declaração de pacote é válida, o restante deve ser feito via importação. O uso de pacotes externos (pacotes próprios ou de terceiros) na linguagem Java tem a seguinte sintaxe:

```
package edu.org.meuspacotes.*;        // Tudo de meuspacotes.
package edu.org.meuspacotes.CPF;      // Uma classe específica.

package com.application.collaborate.server.*;
package com.application.collaborate.client.*;
```

Qual é a vantagem de usar esses pacotes? Simples. Se for pacote da linguagem Java servirá para você poder programar senão, não tem como, e se for pacotes de terceiros, servirá para que você possa usar uma implementação existente no pacote para um objetivo específico.

Você também pode desenvolver seus próprios pacotes, mas para isso o seu conhecimento em ambiente Java precisa estar bem amadurecido. A vantagem disso é que poder redistribuí-lo compartilhando assim seus conhecimentos e trocando experiência.

Quando usamos uma IDE, ou seja, um ambiente de desenvolvimento de aplicativos como *JBuilder*, Eclipse ou *NetBeans* por exemplo, elas usam a regra *package <diretório_da_aplicação>* a ser desenvolvida. Veja isso através dos fontes de sua aplicação e perceba que todos os demais programas (classes) dessa aplicação estarão fazendo referência a esse diretório.

NOTA: Em uma aplicação, por arquivo de classe apenas uma declaração package pode existir, pois na verdade trata-se de um diretório de onde vão ser buscadas as classes. Também é possível colocá-lo como declaração import em alguns casos.

Tabela 1.1: Principais pacotes da linguagem Java.

Pacote	Descrição
java.awt.*	Classes responsáveis pela criação de aplicações gráficas usando AWT – Abstract Window Toolkit, ou seja, um kit de ferramentas para construção de aplicativos baseados em janelas.
java.beans.*	Classes para criação de componentes reutilizáveis, os famosos Java Beans.
java.io.*	Classes responsáveis pelo tratamento de controle de fluxo de dados de entrada e saída.
java.lang.*	Classes e métodos responsáveis usados na programação do dia–a-dia da linguagem Java.
java.math.*	Classe responsável por métodos matemáticos usados na programação.
java.net.*	Classes responsáveis pelo acesso e trabalho através da rede.
java.rmi.*	Classes responsáveis por métodos de execução remota, os Java RMI.
java.security.*	Classes e interfaces responsáveis por métodos de controlam o aspecto sobre segurança no sistema.
java.sql.*	Classes responsáveis por métodos de acesso a banco de dados usando a linguagem SQL.
java.text.*	Classes para métodos relacionados a formatação e customização de formatos textos como data, hora, números, streams etc.
java.util.*	Classe de coleções de framework, modelos de eventos, internacionalização e métodos de geração de números aleatórios, arrays e string tokeninzer.
javax.crypto.*	Classe responsável por métodos para tratamento de criptografia em Java.
javax.imageio.*	Classes e métodos responsáveis para manipulação de imagens, como entrada, saída, tratamento e exibição.
javax.naming.*	Classes e interfaces para acesso ao naming service em Java (JNDI).
javax.net.*	Classes aplicações de rede.
javax.print.*	Classes e métodos para serviços de impressão.
javax.security.*	Provê um framework (camada) para controle autenticação e autorização.
javax.sound.*	Classes e métodos para trabalhar com multimídia.
javax.swing.*	Classes e métodos para desenvolvimento de aplicações gráficas mais modernas e elaboradas que AWT.
javax.xml.*	Classes e métodos para processamento em XML.
org.omg.*	Provê um mapeamento para API Corba am Java.
org.w3c.*	Provê uma interface para DOM que é componente API para Java XML.
org.xml.*	Contém classes e métodos para processamento em XML.

Como podemos ver, esses são os pacotes básicos da linguagem Java e acredite tem MUITA coisa para aprender com eles. Poderemos desenvolver qualquer aplicação com esses pacotes, a não ser que precisemos de um pacote muito específico.

Comentários em código Java

Como já sabemos, é muito importante que na medida em que escrevemos nossos códigos façamos uso da inserção de comentários para que o código fique mais legível para nós mesmos e para outras pessoas que poderão estar lendo nossos códigos. Na linguagem Java podemos fazer isso de três modos distintos. Os comentários em Java tanto servem para tornar nossos programas mais elegantes quanto para gerar documentação e se quiser saber como fazer isso aconselho-o a estudar um pouco mais sobre o utilitário *javadoc* o qual acompanha todos os kits Java.

Faça uso deles e vai perceber o quanto eles são importantes. Se souber usar comentários seus códigos ficarão mais elegantes. Em Java é muito fácil inserir comentários e podemos colocá-los a qualquer momento e para isso veja os exemplos a seguir:

Criar documentação em formato HTML com *javadoc*:

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
```

Semelhante a linguagem C e C++:

```
/* The HelloWorldApp class */
```

```
// The HelloWorldApp class
```

Pronto. Agora não tem mais desculpas para não mais documentar seus códigos. Você poderá acentuar normalmente as palavras em seus comentários, mas não em seus programas textos, pois a saída não será como a esperada a não ser que você use a classe de codificação de códigos de páginas para que a saída de seus programas sejam acentuadas no console texto.

Geralmente não fazemos isso, pois o console só vai servir para testarmos nosso aprendizado de fundamentos de programação, pois quando usamos o modo gráfico a acentuação não apresenta nenhum problema.

Separadores de controle

A linguagem Java oferece um par de seqüências de caracteres de controle de programação unindo-se assim as regras de programação. São separadores simples cuja função é hierarquizar e controlar o fluxo de instruções em um arquivo Java. Esses separadores são:

( ) *Parênteses*: Tem como função conter listas de parâmetros em uma definição e chamada de métodos. Também podemos utilizá-los para definir precedência em expressões, controle de fluxo e para efetuar conversões (*casting*) em tipos primitivos de dados;

{ } *Chaves*: Usamos para trabalhar e inicializar nossas matrizes/vetores/arrays. Também usamos para definição blocos de códigos, classes e métodos;

[ ] *Colchetes*: Usamos para declaração de matrizes e também usamos quando estamos fazendo referência aos valores e matrizes e para controle de índice de argumentos passados via linha de comando;

<> *Maior e Menor*: Usando no Java 5 para definição de tipos genéricos, um novo conceito muito interessante e ágil para armazenamentos de objetos quaisquer;

";" *Ponto e vírgula*: Usamos para terminar uma linha de instrução em nossos arquivos Java;

"," *Vírgula*: Usamos para separação de identificadores em uma declaração de variáveis. Também usamos dentro um laço for do mesmo modo que usamos na linguagem C/C++;

"." *Ponto*: Usamos para separar nomes de pacotes de terceiros e também de pacotes internos e usamos também para identificar métodos externos de classes externas na atribuição de ação de objetos.

Convenções de nomenclatura na codificação

A convenção proposta pela *SUN Microsystem* para uma codificação correta de códigos em linguagem Java é simples de entender, basta que você compreenda os 5 (cinco) principais pontos chave de uma codificação com relação aos seus nomes. Acompanhe:

1. *Classes*: Devem ser palavras completas e devem fazer uso de letras minúsculas e maiúsculas sendo que toda classe começa com letra maiúscula;
2. *Interfaces*: Deverão seguir a mesma regra das classes;
3. *Métodos*: Deverão fazer uso de letras minúsculas e maiúsculas sendo que sempre deverão começar com letra minúscula e se forem formados pela composição de mais de uma palavra a primeira é em minúscula e as seguintes têm suas iniciais em maiúsculas;
4. *Variáveis*: As variáveis de instância e de classe seguem a mesma regra aplicada aos métodos de classe. São aceitos caracteres normais e $ e _ (sublinhado), também é válido números exceto no início e espaços são inválidos;
5. Nomes das variáveis constantes: Deverão sempre ser declaradas em letras maiúsculas evitando o uso de constantes internas a linguagem e se forem constantes compostas usam-se o caractere "_" para facilitar a depuração;
6. Para nomes de pacotes (coleções de classes): Segue a padronização semelhante a URL como veremos no capítulo 10 do estudo sobre orientação a objetos.

Tipos de aplicações com Java

Eu acredito que nessa altura você já saiba que tipo de aplicativos poderá desenvolver em Java. Sim, podemos desenvolver qualquer aplicativo como fazemos em *Visual Basic* e *Borland Delphi*, tudo é questão de jeito, experiência, prática e planejamento.

Nós podemos desenvolver uma aplicação para o modo texto, para o modo gráfico usando Java AWT ou *Swing* e podemos desenvolver aplicação para a Internet através do uso de Java Applets. O único cuidado quando estamos desenvolvendo aplicação em Java Applets é que os *applets* apresentam restrições quanto ao local em que estão sendo executado.

Se você for desenvolver uma aplicação que faça uso de acesso a disco (escrita/leitura), por exemplo, os *applets* não são recomendados, pois remotamente eles não têm esse direito. O lógico seria fazer então uma aplicação normal, mas que pudesse ser executada remotamente com propriedades de leitura e escrita local.

Os tipos de aplicações que podemos escrever em Java são:

1. Aplicações em modo texto e aplicações em modo gráfico usando *AWT* e/ou *Swing*;
2. Aplicações gráficas que rodam dentro do navegador usando *applets*;
3. Aplicações "embarcadas" (*Embeded Applications*), ou seja, são as aplicações que você encontra nos celulares, palms, alguns relógios e outros dispositivos portáteis.

Como escrever um programa Java

Escrever programas ou projetar sistema em Java quer uma série de procedimentos para que o resultado final siga uma padronização profissional. Pois isso requer um monte de conhecimento desde as ferramentas de modelagem de dados, UML, IDE e tipo de programação que vai ser usada, ou seja, se vai ser uma aplicação de console, uma aplicação gráfica usando *AWT/Swing*, se vai ser uma aplicação usando Java Applets, *Servlets* ou mesma se vai ser uma aplicação para celular usando J2ME.

Siga esses procedimentos e vai ser muito simples pelo menos saber o que fazer, o que usar e como usar o que dependerá dos seus conhecimentos dessas ferramentas:

1. Aplicação é local, remota ou para micro dispositivos?
2. Usa base dados? Se usa, qual é a base?
3. Precisa de modelagem para a base de dados?
4. Precisa de modelagem para a aplicação?
5. Usa alguma IDE? Se usa, qual a IDE oficial?
6. É necessario usar uma análise de sistema?
7. ...

Bom, se você prestou atenção, somente nesses seis itens existe conhecimento que não acaba mais. A outra pergunta é: — O quanto disso aí você sabe ou domina? Pois é, sorte sua, pois geralmente em desenvolvimento Java estamos participando de uma equipe onde cada um tem uma tarefa a ser trabalhada, portanto ache logo a sua posição nessa equipe. ☺

Programação braçal versus visual

Agora nós chegamos em um ponto que com certeza você já está se perguntando há um bom tempo e está cheio de dúvidas:

"— Devo programar no braço eu devo usar uma aplicação para me ajudar? No braço pode demorar muito para o que eu quero fazer."

Bom, em primeiro lugar eu vou lhe dar umas dicas muito preciosas para encontrar o melhor caminho. Primeiro aprenda o fundamento da linguagem Java e para isso nada melhor que um pequeno editor de códigos Java. Para Windows existem vários editores de uso comercial, *shareware* e *freeware* (gratuito), para o ambiente Linux também.

No Linux temos o editor *kate* e o *kwrite/kedit/gedit* que já vem com a maioria das distribuições mais recentes e o mais interessante e como não poderia deixar de ver o bom e velho editor VI que faz a colorização automática do código.

É lógico que você não vai querer desenvolver todo um sistema complexo na mão. No entanto, quando começar a fazer seus programas pequenos "no braço" isso vai dar bastante base para quando usar uma IDE a sua preocupação vai ser como aprender a usar a IDE, pois o restante já sabe como fazer, entendeu? Sendo assim, nada melhor do que começar a fazer as coisas na mão mesmo.

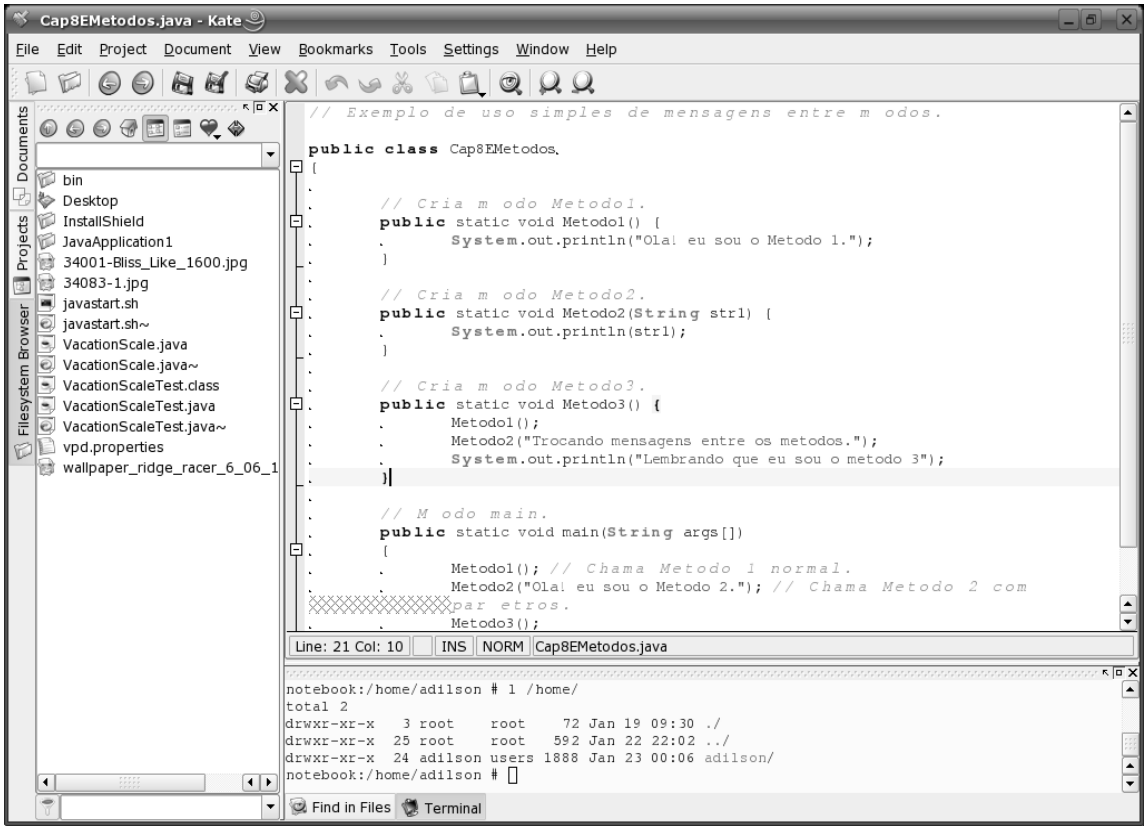

Figura 1.3: Sistema Linux, editor kate em ação.

Se souber projetar, até mesmo uma aplicação simples conseguirá fazer usando um editor simples, acredite. Mas pode escolher qualquer um que achar melhor, e para um primeiro momento a digitação pura do código é melhor, pois assim você vai ficando mais íntimo da linguagem e vai aprendendo como que funciona a filosofia da programação em linguagem Java. Então, você percebeu que o fundamento do Java vai ser mesmo no braço, isso é bom porque você vai realmente aprender Java sem os vícios de um ambiente integrado de desenvolvimento (IDE).

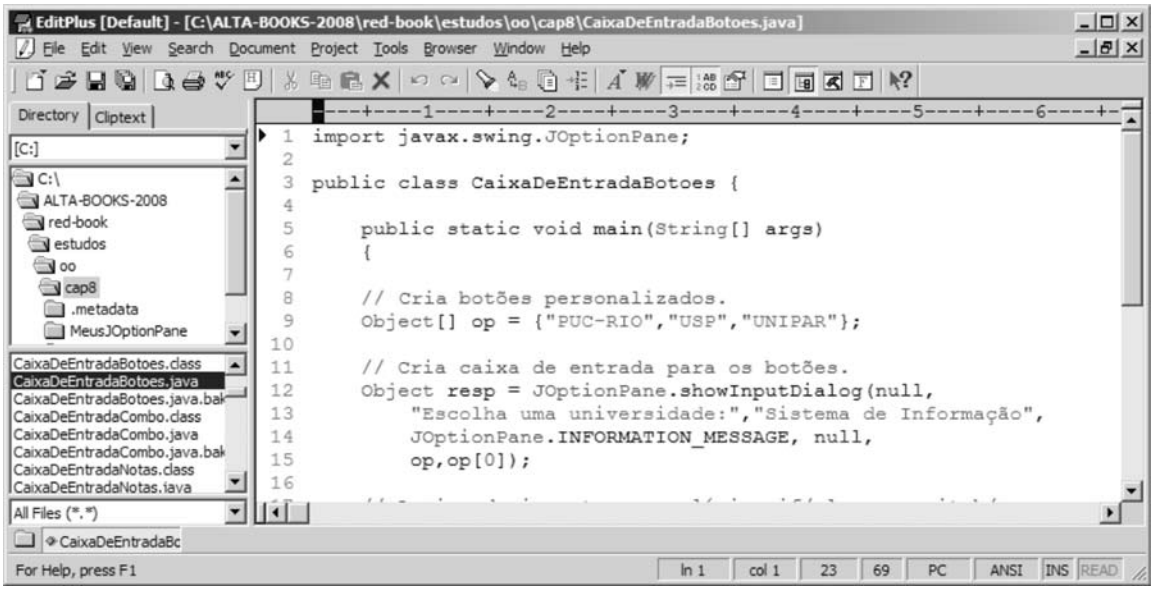

Figura 1.3.1: Editor EditPlus (ocupa apenas um disquete).

Depois que estiver bastante familiarizado como código Java no braço mesmo, aí então eu lhe aconselho a usar uma IDE para seu desenvolvimento profissional e de suas aplicações, mas isso também vai depender do que você quer fazer e com que vai trabalhar.

No sistema Windows tem um aplicativo bem simples e muito "bonzinho" de usar que é *Editplus* (www.editplus.com). Ele é tão pequeno que cabe em um disquete, também faz colorização automática do código Java, além de outros códigos. Eu o recomendo para que você se acostume com o estilo de programação em Java e além do mais ele é um *shareware* que não expira, ele apenas fica informando que o tempo de uso já venceu, mas permite que mesmo assim você o continue usando.

Se você for trabalhar com aplicações corporativas e banco de dados *Oracle* eu lhe aconselho o *JDeveloper 10g* (www.oracle.com) que já faz toda a integração entre parte servidora, parte cliente e parte interface. Mas eu já lhe digo, prepare seu bolso e sua máquina, pois embora a primeira vista ele seja fácil de entender vai precisar fazer um curso sobre como usar o *JDeveloper com* bancos *Oracle*, os cursos não são nada baratos, embora o aplicativo seja gratuito digamos assim.

E prepare sua máquina, pois eu lhe aconselho no mínimo uns 512 MB só para começar a brincar, uns 768 MB já da para ter uma rotina normal de trabalho, mas se quiser abusar pode dispor de 1 GB de RAM em sua máquina que você realmente vai desenvolver aplicações corporativas.

Você poderá usar o *JBuilder* da *Borland* (www.borland.com) para aplicações não tão robustas e pesadas pois a curva de aprendizado do *JBuilder* é menor que a do *JDeveloper* e você consegue fazer coisas realmente com qualidade usando o *JBuilder* nas versões mais recentes.

Você também poderá usar o *Eclipse* (www.eclipse.org) que é o que a maioria dos usuários Java acaba aprendendo. O projeto *Eclipse* está bem maduro e esse *framework* de desenvolvimento Java a cada dia fica mais completo e robusto.

O *NetBeans* (www.netbeans.org) além de ser de uso livre é muito bom e também apresenta uma curva de aprendizado pequena em relação aos outros e ainda vem incluso nas últimas versões do kit de desenvolvimento do Java distribuído pela SUN. Sua interface passou por uma remodelagem para que ficasse mais intuitiva e fácil para o usuário. É muito interessante desenvolver usando o *NetBeans,* pois com ele podemos criar aplicações completas.

O *NetBeans* nas ultimas versões como 5/6 ou superior trás uma excelente idéia chamada de *Matisse* que é um *free layout* auto ajustável, o desenho de formulários dá-se de modo livre como se fosse em Delphi, mas quando um componente chega próximo ao outro ou na borda do formulário automaticamente aparecerem guias de ajustes precisos para posicionamento do componente, isso era o que estava há tempos faltando em uma IDE de alto nível.

No entanto eu lhe aconselho que antes que você opte por alguma dessas IDE's que procure se identificar com a faixa de mercado que está atuando ou trabalhando e depois procure se familiarizar com cada uma delas, por exemplo, tire uma semana para cada uma e nessa semana explore o máximo que puder.

No entanto, se tem tempo para estudar as várias IDE's Java que tem por aí, procure na Internet que você vai achar excelentes ferramentas para desenvolver em Java, tudo é uma questão de tempo.

Meu primeiro programa: Olá Mundo!

Acredito que você percebeu que já apareceram alguns exemplos de programas no decorrer da leitura deste capítulo, mas ainda não havíamos feito os nossos programas, pois os outros eram apenas exemplos.

Primeiramente para gravar os nossos exercícios do livro eu sugiro que se crie um diretório com o nome de *estudos*. Em MS-DOS use o comando **md c:\estudos** e depois trabalhe dentro desse diretório ou podemos criar esse diretório/pasta pelo *Windows Explorer*.

Para fazer isso usando o sistema Linux, também é simples, use o comando **mkdir estudos**, não se preocupe com o local, se estiver conectado como usuário *root* (administrador) esse diretório ficará em */root/estudos* e se estiver conectado como usuário normal, o diretório ficará dentro do local dos usuários comuns que é o */home* ficando assim */home/adilson/estudos*, por exemplo, se for para o usuário *adilson*.

OK! Chega de conversa vamos criar nosso primeiro programa em Java, ele vai ser bem simples seguiremos o exemplo do *HelloWord.java* porém vamos chamá-lo de *OlaMundo.java*. Se estiver usando o Windows poderá usar o bloco de notas e quando salvar, salve como o nome desse exemplo e no tipo do arquivo escolha *, assim ele será gravado em formato texto puro.

Mas se tiver usando o sistema Linux verifique se o utilitário *kate* está instalado, pois ele colore automaticamente os códigos em Java. Se não tiver, use o *kedit*, *kwrite* ou *gedit* que já vai ajudar bastante. Mas se você for bem radical use o editor vi, apesar de ser em modo texto esse também faz a colorização automaticamente dos códigos.

DICA: Defina bem a estrutura de seus programas em Java e evite usar arquivos muito grandes. Não é interessante usar um arquivo com mais de 2.000 (dois mil) linhas, se isso acontecer tente dividi-lo em classes para melhor gerenciamento.

Se estiver usando o Linux e for usar o editor *vi* use o usando o comando *vi OlaMundo.java* para criar seu arquivo e depois de digitado aperte a seqüência de teclas *ESC:wq!*. Assim você salva e sai do editor.

```
/** Programa…: OlaMundo.java */
/*  Objetivo…: Mostrar a frase Ola Mundo! Na tela */
//  Compilar…: javac -verbose OlaMundo.java
//  Executar…: java OlaMundo

public class OlaMundo {
      public static void main(String[] args)  {
         System.out.println("Ola Mundo!");
      }
}// Fim do programa.
```

Tome cuidado para não errar na digitação e a frase "Ola Mundo!" é entre aspas duplas e por enquanto sem acento, pois você ainda não aprendeu a colocar acentuação em Java para o modo texto, pois sai uns caracteres estranhos. Para o modo gráfico a acentuação é normal.

Lembre-se: As chaves não precisam necessariamente estar na mesma linha da declaração da classe e do método principal, isso é uma questão de gosto e de visualização, no decorrer dos exemplos do livro você verá dos dois jeitos, ou seja, abaixo da declaração e junto com as declarações de classe e de métodos.

```
/** Programa: OlaMundo2.java */

public class OlaMundo2 {
      public static void main(String[] args)        {
         System.out.println("Ola Mundo!\n--------------------------------");
         System.out.println("Aprendendo programar em JAVA....");
      }
}// Fim do programa.
```

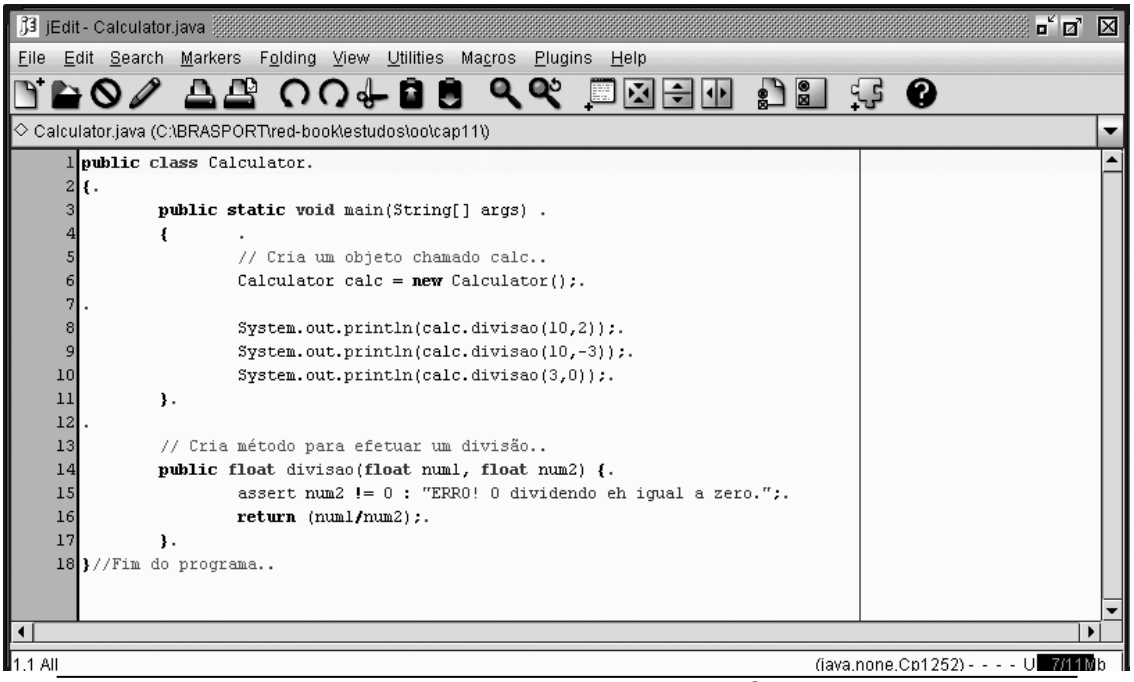

Figura 1.4: Jedit (Windows), roda também no Linux.

Há outro editor interessante que é o Jedit (www.jedit.org), você pode baixar a versão para MS Windows e rodar também no sistema Linux, pois ele tem a extensão "*.jar*". Para executar no Linux é simples, use o comando *java –jar /...caminho.../jedit.jar*. Onde caminho é o diretório que você o instalou lá no Windows. Eu gostei dele simplesmente por esse motivo.

Como compilar e executar um programa Java

A primeira coisa que precisamos saber é que o nome do arquivo a ser salvo DEVE ser o mesmo nome da classe principal se essa for uma classe pública. É a classe principal que informa o nome da aplicação, se sua classe principal tem um nome e seu arquivo salvo tem outro, mesmo que todo o seu programa esteja correto não vai conseguir compilar. Acompanhe:

```
# javac OiMundo.java
OiMundo.java:3: class OlaMundo is public, should be declared in a file named OlaMundo.java
public class OlaMundo
       ^
1 error
```

No exemplo anterior o nome do arquivo é *OiMundo.java* mas ele possui uma classe principal que se chama *OlaMundo*, então, por convenção devemos gravar o arquivo como *OlaMundo.java* senão teremos erro de compilação.

Bom, após digitar um programa e verificar que não tem nenhum erro de digitação, sei que está ansioso para ver esse negócio funcionado. A compilação de um programa Java segue a seguinte sintaxe:

```
# javac opções arquivo.java
```

...e no momento devemos estar dentro do diretório onde iremos compilar nosso programa. É possível estar fora do diretório de compilação ou então informar para onde é que vão parar os arquivos *.class* que são os arquivos fonte *.java* depois de compilados. Calma lá. Se olharmos no nosso diretório poderemos verificar que o nosso arquivo *OlaMundo.java* está prontinho esperando para ser compilado.

Para compilar o arquivo use o comando *javac OlaMundo.java*, e se tudo ocorreu bem, agora existe mais um arquivo chamado *OlaMundo.class*, esse é o arquivo que foi compilado e agora vai ser interpretado (executado) pela JVM.

Para executar agora o seu programa apenas use o comando *java OlaMundo* (sem o sufixo *.class*), automaticamente será executado o arquivo *.class* que é o arquivo compilado. Viu só que coisa mais simples do mundo. Se desejar ver mais detalhes sobre como a compilação está funcionando use o comando *javac –verbose OlaMundo.java*, assim veremos o compilador passando por todas as classes internas do Java necessárias para o funcionamento do seu programa.

```
C:\estudos>javac -verbose OlaMundo.java
[parsing started OlaMundo.java]
[parsing completed 171ms]
[search path for source files: [., C:\Arquivos de programas\QuickTime\QTSystem\Q
TJava.zip]]
[search path for class files: [C:\Arquivos de programas\Java\jdk1.5.0_10\jre\lib
\rt.jar, C:\Arquivos de programas\Java\jdk1.5.0_10\jre\lib\jsse.jar, C:\Arquivos
 de programas\Java\jdk1.5.0_10\jre\lib\jce.jar, C:\Arquivos de programas\Java\jd
k1.5.0_10\jre\lib\charsets.jar,...
...
[wrote OlaMundo.class]
[total 2781ms]
```

Você pode compilar qualquer programa fonte .java mas somente poderá executar o programa que possuir o método *main()* e que haverá apenas um por programa fonte, pois o restante serão usados como instâncias e bibliotecas.

Se tentar executar o programa sem o método *main()* será lançada uma exceção em tempo de *run-time*:

```
# java OlaMundoSemMain
Exception in thread "main" java.lang.NoSuchMethodError: main
```

O compilador Java oferece algumas outras opções para você compilar seus programas. Essas opções estão explicadas na tabela seguinte.

Tabela 1.2: Quadro de opções (algumas) de compilação.

Opção	Descrição
-classpath caminho	Determina o *path* (caminho) no qual o compilador vai procurar por classes.
-d diretório	Determina o diretório no qual o compilador armazenará os arquivos *.class* de saída de compilação.
-g	Informa ao compilador para criar o arquivo de informação para *debug*.
-nowarn	Informa ao compilador para não mostrar quando da compilação de um arquivo.
-O	Informa ao compilador para aperfeiçoar a compilação de um programa.
-verbose	Muito útil, pois informa ao compilador para exibir as informações sobre os procedimentos de compilação.

Mas se por algum motivo precisarmos compilar para ser usado com a JVM 1.4 estando o sistema com a JVM 1.5, usamos a seguinte linha de comando (O diretório das bibliotecas para JVM 1.4 deve estar no sistema).

```
# javac -target 1.4 -bootclasspath jdk1.4.2/lib/classes.zip -extdirs "" ArqAntigo.java
```

Acompanhe aqui alguns exemplos. Informando ao compilador para incluir a classe *classes.zip* no arquivo *Teste.java*. A importância disso é que podemos fazer reaproveitamento de códigos, por exemplo, vamos supor que você criou uma classe para verificar CPF, RG, CNPJ, etc.

```
# javac -classpath c:\java\lib\classes.zip;c:\classes Teste.java
```

Informando ao compilador para incluir os arquivos compilados (de saída) *.class* no diretório denominado compilados, desse modo todos os seus executáveis interpretados estarão em um único lugar.

```
# javac -d c:\classes\compilados applet.java
```

Usando essa idéia poderíamos ter a seguinte situação, um diretório chamado *sources* para os fontes e outro chamado *classes* para os *ByteCodes*. Veja o exemplo para Windows e Linux:

```
C:\>estudos\classes\
C:\>estudos\sources\
```

```
/opt/estudos/classes/
/opt/estudos/sources/
```

Para saber a versão instalada do JSDK em seu computador:

```
# java -version
java version "1.5.0_10"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_10-b03)
Java HotSpot(TM) Client VM (build 1.5.0_10-b03, mixed mode, sharing)
```

Se você fez um *update* para a mais recente versão do Java terá esse resultado:

```
C:\Documents and Settins\adilson>java -version
java version "1.7.0-ea"
Java(TM) SE Runtime Environment (build 1.7.0-ea-b23)
Java HotSpot(TM) Client VM (build 12.0-b01, mixed mode, sharing)
```

Geralmente, um aplicativo *stand-alone* (chamado via linha de comando ou atalho na área de trabalho) é composto por várias classes. Como cada classe é "compilada" para um arquivo *.class* distinto, é necessário saber qual classe possui o método estático *main(String args[])* e passar o nome dessa classe como parâmetro para o programa java, sem a extensão .class.

Veja os exemplos de como executar uma aplicação em java.

```
java NomeDaClasse
java pacote.subpacote.NomeDaClasse
java -classpath c:\classes;c:\bin;. pacote.Classe
java -cp c:\classes;c:\bin;. pacote.Classe
java -cp %CLASSPATH%;c:\mais pacote.Classe
java -cp biblioteca.jar pacote.Classe
java -jar executavel.jar
```

Para rodar aplicações gráficas, use *javaw* (Windows):

```
javaw -jar executavel.jar
javaw -cp aplicacao.jar;gui.jar
```

DICA: O comando java quando executado de dentro de um terminal não permite a digitação de comandos neste terminal, porém o comando javaw carrega o programa java e permite que comandos sejam usados no terminal.

Muitas vezes, as classes Java podem ser agrupadas em pacotes (diretórios) que podem estar ou não compactadas. Se o programa Java estiver dentro de um pacote não compactado, deve-se incluir o nome do pacote no parâmetro passado para o comando *java*, estando fora do pacote (o diretório atual deve ser o diretório que contém o pacote). Por exemplo, o comando a seguir executa o aplicativo *TesteDeClasse*, contido no diretório *minhaClasse*.

```
# java minhaClasse.TesteDeClasse
```

Para executar um aplicativo Java *stand-alone*, é necessário ter o JRE – *Java Runtime Environment* instalado na máquina, que pode ser baixado da Internet na própria página da SUN. É possível compilar o aplicativo em Java de modo normal para modo nativo em Windows, gerando assim um executável *.exe*, mas mesmo que o aplicativo seja compilado em modo nativo é necessário ter a JRE.

NOTA: O propósito de se criar uma aplicação "nativa" em Windows ou Linux é a execução mais rápida, mas isso comprometerá a portabilidade da aplicação e uma vez nativo, a aplicação será nativa no sistema operacional onde foi compilada.

Figura 1.5: Sem JRE aplicações em Java (.exe e .jar) não funcionam.

Se o programa Java estiver dentro de um pacote compactado (.zip ou .jar), deve-se usar um comando como o que está a seguir, passando o parâmetro –jar tanto para arquivos .jar quanto para arquivos .zip:

```
# java -jar TesteDeClasse.jar ou
# java -jar TesteDeClasse.zip
```

Há uma outra maneira de compilar seus programas em Java. Quando nós usamos rotinas de controles de erros como *try/exceptions* ou *assertions* podemos ter um maior controle sobre futuros erros em nossos programas. Quando fazemos uso de *assertions* o que geralmente é usado quando já temos uma boa base sobre controle de erros é possível primeiramente verificar se possíveis erros podem ocorrer. Através do pacote *java.lang.AssertionError* é que podemos através da palavra reservada *assert* inserir esse tipo de controle em nosso código.

O uso de *assertions* também é conhecido como uso de assertivas, informa que mesmo que um código tenha sido compilado com esse tipo de uso, quando da hora da compilação você deverá explicitamente forçar a sua utilização, pois por padrão as assertivas não são ativadas.

As assertivas não são compiladas nem executadas por padrão e é possível instruir a JVM desativar assertivas para uma classe e ativar para um dado pacote em uma mesma linha de comando. Mas, não se preocupe com isso agora, pois primeiramente precisamos aprender os fundamentos da programação e lá na Parte 2 do nosso estudo veremos como tudo isso funciona. Para habilitar ou desabilitar o uso das assertivas em seus programas veja os comandos a seguir:

1. *java -ea* ou *-enableassertions*: Habilita o uso de *assertion*;
2. *java -da* ou *-disableassertions*: Desabilita o uso de *assertion*;
3. *java -ea:br.com.unipar*: Habilita para o pacote *br.com.unipar*;
4. *java -ea -das*: Habilita em âmbito geral e desabilita nas classes do sistema;
5. *java -esa* ou *-enablesystemassertions*: Habilita em todas as classes do sistema;

Uma idéia legal que acontece com a linguagem Java e a qual eu achei muito interessante é a compilação *on-line*, ou seja, você manda seu código fonte para uma página da Internet e lá um sistema se encarrega de compilar o seu código. Isso é interessante se você não tem espaço em sua máquina para instalar um J2SE ou J2EE completo, ou então quer fazer isso para seus estudos, pois você poderia estar em um *cyber café* ou *lan house* e nessas máquinas não ter nenhum ambiente Java.

Hoje já temos uma boa medida de páginas com esse propósito, algumas são livres (uso gratuito), porém outras cobram um taxa para fornecimento do serviço, basta efetuar uma pesquisa no *google* e encontrará várias páginas com esse propósito.

Com certeza quando da digitação dos nossos primeiros programas em Java, sempre erramos, porém, até pegarmos o jeito percebemos que o compilador Java mostra erros no momento da compilação, mas em uma outra hora o programa parece estar certo e mesmo assim aparecem erros.

Como identificar esses erros? Bom, nós temos praticamente dois tipos de erros em Java:

1. *Compiler Error*: Esse é um dos erros mais comuns, ou seja, no momento da compilação o compilador percebeu um erro no código e não pode prosseguir com a compilação e retorna um erro. Exemplo: Método não estático querendo acessar um atributo com visibilidade ou acesso restrito a esse tipo de método;
2. *Run-Time Error*: Esse também é um erro comum. Aqui o compilador passa batido, mas quando da execução do programa é interrompido abruptamente e para. Exemplo: Uma divisão por 0 (zero).

Bom, isso é o básico que deveremos saber por enquanto, claro que vamos nos aprofundar nesses conhecimentos no decorrer da leitura e exercícios do capítulo. Agora já pode falar que você está esperto na compilação de programas em Java, certo? Com um pouco mais de prática poderá melhorar ainda mais, por exemplo, mudar a cara de uma aplicação gráfica passando argumentos via linha de comando. Assim uma aplicação gráfica poderá ter a cara do Windows, do Motif, do Java padrão, do Mac OS etc.

Quer saber como? Ainda não é hora, mas dê uma olhada no programa *LookAndFeel.java* que você vai entender do que eu estou lhe falando, observe:

```
 // Isso é semelhante aos skins de alguns programas.
import javax.swing.*; // Importa pacote gráfico.
import java.awt.*;

public class LookAndFeel {
      public static void main (String[] args)       {
         // Bloco para escolha do Look And Feel.
         Try { UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
         } catch (Exception e){}

// Modelo para CDE/Motif.
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

// Modelo para MS Windows.
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

// Modelo para Mac OS X.
UIManager.setLookAndFeel("com.sun.java.swing.plaf.mac.MacLookAndFeel");

//...e para o sistema operacional que estiver rodando a aplicação...
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
*/
      JFrame janela = new JFrame (":: Usando skins em Java"); // Janela aplicação.
      janela.setSize(500,70);

      // A janela pode ser fechada pelo botão X.
      janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // Cria um objeto container associado ao frame.
      Container c = janela.getContentPane();
      c.setLayout(new FlowLayout());

      // Adicionar botões no objeto container.
```

```
      c.add(new JButton("Cadastrar"));
      c.add(new Jbutton("Alterar"));
      c.add(new JButton("Pesquisar"));
      c.add(new Jbutton("Apagar"));
      c.add(new Jbutton("Imprimir"));

      // Mostrar o Frame completo na tela.
      janela.setVisible(true);
      }
}// Fim do programa.
```

Figura 1.6: Resultado do programa com uso de skin padrão.

Para que você possa mudar o *skin* (pele) de sua aplicação é só substituir no bloco *try* por alguma linha do comentário. Faça isso e acompanhe o resultado. A seguinte figura tem com o *Look And Feel* do Mac:

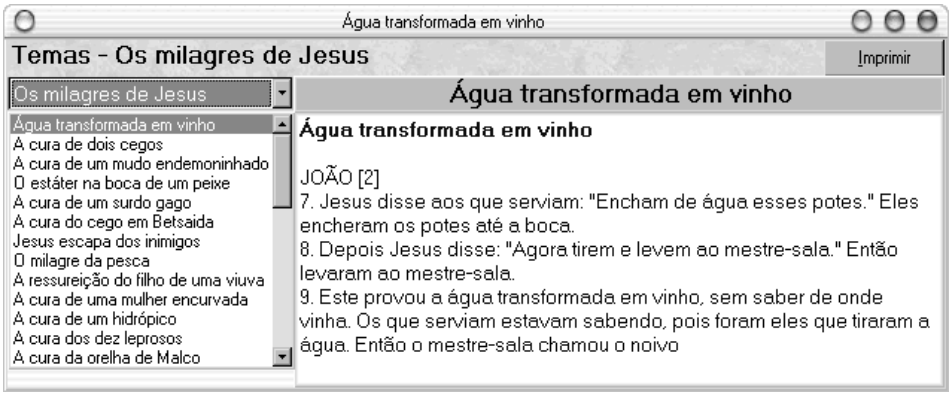

Figura 1.6.1: Uma aplicação com cara de Mac OS X.

Para ter uma idéia da importância desses recursos, imagine que está usando o ambiente Windows e pode desenvolver uma aplicação com a cara do Gnome do sistema Linux conforme a figura seguinte.

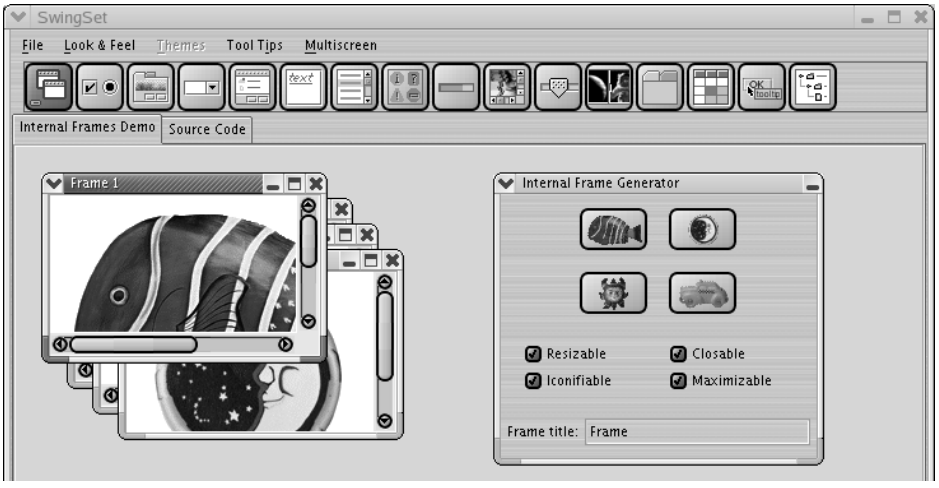

Figura 1.7: Uma aplicação com a cara do gerenciador de janelas Gnome.

O Borland Delphi tem uns componentes de terceiros que também fazem isso e um dos mais famosos é o VCLSkin e para Java existem vários, um mais lindo que o outro. Mas esses *skins* podem ser simples ou não de adicionarmos em nossos códigos, cada um de um jeito. O ponto de partida para inúmeros links é www.javootoo.com lá você encontra até uma pré visualização de cada um deles. Escolha o melhor.

Como gerar a documentação

Algo muito importante que precisamos saber é como gerar a documentação padronizada dos nossos futuros sistemas. Agora no momento em que nossos programas estão pequenos parece que não há sentido. Mas imagine que você tenha que dar manutenção em um sistema grande. Se tivesse uma documentação sobre o sistema seria muito mais importante, pois através dela é mais fácil estudar o sistema em si e saber o que ele faz, ou seja, o que cada classe, método faz no sistema.

Para gerar a documentação dos nossos sistemas utilizamos o *javadoc* que é uma ferramenta para gerar documentação da API a partir dos comentários no código. Esses comentários são escritos usando marcações simples (*tags*) definidas na sintaxe Java.

Entre elas, as mais utilizadas são:

1. *@author*: Define o autor da classe;
2. *@version*: Define a versão da classe;
3. *@param*: Define um parâmetro de um método;
4. *@return*: Define o retorno do método;
5. *@throws* (ou *@exception*): Indica as exceções tratadas ao chamar o método.

A documentação é produzida no formato HTML e muito embora ensinar como documentar um projeto não seja o escopo deste livro vamos fazer um pequeno exercício e gerar a documentação para ele. A documentação gerada é idêntica a API do Java, assim, você se acostuma com as duas coisas.

O primeiro passo é lembrar que a linguagem Java tem um tipo especial de comentário que é semelhante ao da Linguagem C/C++, tudo que estiver dentro desses comentários será usado para gerar sua documentação então se quisermos que na documentação apareça a informação do que a classe faz ou a importância de um método e o que ele faz dentro do sistema, nos colocamos esses tipos de comentários acima das classes e dos métodos que queremos gerar esse tipo de informação, veja:

```
/** Isso é um comentário para o Javadoc em uma linha. */

/** Isso é um comentário para o Javadoc
em duas linhas. */

/** Isso é um comentário para o <b>Javadoc</b>
em duas linhas e com tags em HTML. */
```

Interessante saber é que podemos também usar as *tags* da linguagem HTML para melhorar o visual da documentação, mas cuidado para não fugir do padrão. Há também *tags* especificas para o utilitário *javadoc* que é usado para gerar a documentação.

Bom, se você entendeu, analisaremos o programa *AloMundoDoc.java* que é bem simples e servirá para nos dar uma base de como gerar a documentação.

```
/** <b>AloMundo - Documentação</b>
* @see java.lang.Object
* @author Adilson Bonan
*/
// Cria classe pública para o programa.
public class AloMundoDoc {
      /** Texto que vai ser exibido. */
      static String msg = "Criação de documentação para programa em Java.";
      /** Executa implicitamente o método construtor.       */
      AloMundoDoc(){}
      /** Cria um método público para mostrar a mensagem. <br>
          Esse método recebe uma variável de classe como parâmetro.
          @param msg é uma String a ser passada.
      */
      public static void Alo(String msg) {
         System.out.println(msg);
      }
      /** Método principal do programa.*/
      public static void main(String[] args){
         Alo(msg);
      }
} // Fim do programa.
```

Agora para gerar a documentação correta usamos um utilitário chamado *javadoc* com o parâmetro *-verbose* para verificarmos por onde ela passou nos pacotes e *-author* para que conste na documentação o nome do autor do programa assim:

```
C:\estudos\oo\cap1>javadoc -verbose -author AloMundoDoc.java
Loading source file AloMundoDoc.java...
[parsing started AloMundoDoc.java]
[parsing completed 120ms]
Constructing Javadoc information...
[search path for source files: .,"C:\Arquivos de programas\Java\jdk1.6.0_10"]
[search path for class files: C:\Arquivos de programas\Java\jdk1.6.0_10\jre\lib\
resources.jar,C:\Arquivos de programas\Java\jdk1.6.0_10\jre\lib\rt.jar,C:\Arquiv
os de programas\Java\jdk1.6.0_10\jre\lib\sunrsasign.jar,C:\Arquivos de programas
...
Standard Doclet version 1.6.0_10-rc
Building tree for all the packages and classes...
Generating AloMundoDoc.html...
[loading java\io\Serializable.class(java\io:Serializable.class)]
Generating package-frame.html...
Generating package-summary.html...
Generating package-tree.html...
Generating constant-values.html...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating help-doc.html...
Generating stylesheet.css...
[done in 3054 ms]
```

Percebemos aqui que foi criado uma série de arquivos HTML e o que nos interessa é o *index.html* que pode ser aberto em qualquer navegador.

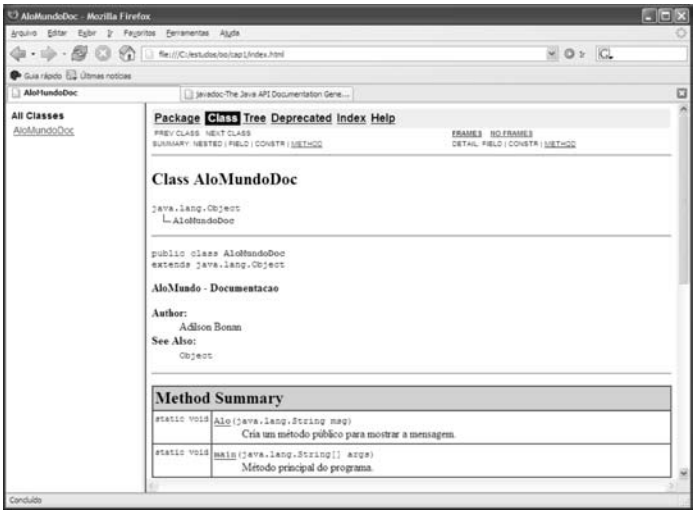

Figura 1.8: A documentação do programa AloMundoDoc.java.

Se for necessário criar a documentação de uma classe qualquer em outro diretório usamos *javadoc MinhaClasse.java -d diretório-destino*. Quando a opção -d não é usada, os arquivos de documentação são criados no diretório corrente.

Para criar a documentação para várias classes basta usar o coringa assim *javadoc *.java*.

Para produzir uma documentação que contém caracteres acentuados devemos usar a opção *-charset*, assim *javadoc -charset "iso-8859-1" MinhaClasse.java*.

Por padrão, o utilitário *javadoc* coloca apenas a documentação associada aos métodos públicos. Para incluir os métodos privados devemos usar a opção *-private*.

Usando passagem de parâmetros

Olha só que interessante apesar de simples essa idéia. Vamos supor que nós quiséssemos passar algo como valores, caracteres ou strings via linha de comando assim como nos comandos do DOS ou do Shell no sistema Linux. O que o programa faz é ler a linha de comando, verificar se aquela opção existe e caso positivo executar tal operação. Caso negativo emitir uma mensagem de erro/aviso ou nenhuma mensagem.

Para entender como funciona a passagem de parâmetros veja parte do corpo da programação a seguir. A leitura da passagem de parâmetros dá-se em (*String[ ] args*) e se você conseguir entender isso, o que é muito fácil, então você conseguirá manipular os parâmetros passados.

```
// Java 4 ou inferior.
public static void main (String[] args){
...
}
```

```
// Java 5 ou superior.
public static void main (String... args){
...
}
```

DICA: O primeiro argumento é args[0], o segundo é args[1] e o enésimo argumento é args[n], ou seja, a linha de comando é um array de strings e se for usar números deve-se usar métodos apropriados de conversão, lembre-se disso.

A checagem do número de parâmetros fornecidos constitui boa programação e verificar esse valor no início da execução e emitir mensagem de erro caso incorreto é uma das preocupações do seu programa. Para passarmos parâmetros para nosso programa é simples, apenas seguimos a seguinte idéia:

```
# java programa parametro1 parametro2 ... parametroN
```

Observações importantes sobre parâmetros:

- Como passar e ler um parâmetro, como passar e ler N parâmetros;
- Como saber o total de parâmetros, como retornar tamanhos de parâmetros;
- Como tratar a passagem de parâmetros;
- Como criar blocos de exceções para parâmetros.

Se dominarmos essa parte, já teremos feito muita coisa, pois isso vai ajudar muito quando estivermos aprendendo algoritmos onde precisaremos passar informações via linha de comando.

Agora vejamos o programa *Args0.java* que é muito simples, pois ele não faz controle de quantos parâmetros foram passados e não usa controle de exceções. Não se esqueça que para compilar use o comando *javac Args0.java* e para executar use *java Args0*.

```
public class Args0 {
      public static void main(String[] args)        {
         System.out.println("Voce digitou: "+args[0]);
      }
}// Fim do programa.
```

Observe a saída desse programa se não passarmos nenhum parâmetro. Será gerado um erro *"ArrayIndexOutOfBoundsException"* e isso por enquanto quer dizer que devemos tratar essa passagem de parâmetro ou com uma declaração *if* ou com blocos *try/catch*. Assim poderemos checar previamente se o usuário forneceu ou não um parâmetro.

```
# java Args0
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at Args0.main(Args0.java:8)
```

Agora veja a saída com dois parâmetros fornecidos, repare que apenas o primeiro foi impresso, pois o *args[0]* imprime somente o primeiro. Viu como é simples.

```
# java Args0 Meu Argumento
Voce digitou: Meu
```